Second Code Improvement Report for the Land Information System Submitted under Task Agreement GSFC-CT-2 Cooperative Agreement Notice (CAN) CAN-00OES-01 Increasing Interoperability and Performance of Grand Challenge Applications in the Earth, Space, Life, and Microgravity Sciences

Version 1.0

History:

Revision	Summary of Changes	Date
1.0	Draft for Milestone G	05/09/04

Contents

List of Figures

1 Description of the Milestone

The milestone G for the Land Information System (LIS) [6, 8] requires the integration of the Land Information System (LIS) with database and visualization functions, and improvement of NOAH [7], VIC [12], and CLM [2] codes within the LIS system to operate at a 1 km horizontal spatial resolution. The improved LIS system will perform global land surface simulation at 1 km resolution with a throughput of approximately $0.4ms/grid{\cdot}day$ of execution time (approximately a factor of 2.5 speedup), on the LIS cluster for near-term retrospective period. The code scaling curves will be presented, and documented source code will be made publicly available via the Web.

2 Summary of LIS Code Improvements

We have greatly improved the LIS code since the first code improvement milestone, to make the system more structured, flexible, efficient, interoperable, and portable. More importantly, the overall performance of LIS has been significantly improved. Specifically, the code improvement for LIS 3.0 covers the following areas:

- Better memory management and utilization patterns;
- More modular design with cleaner implementation of modules;
- Two modes of parallel processing to suit different platforms;
- Customized job management system and distributed IO design;
- More land surface model support with VIC added;
- Faster interpolation routines;
- Partial implementation of ESMF framework.

In this report, we will show detailed performance benchmarking results in Section 6. We will demonstrate that at 1-km resolution, LIS can perform global simulations with a normalized timing of less than $0.1ms/grid{\cdot}day$ (or more than 3 simulated days per day) for all the test cases, surpassing the $0.4ms/grid \cdot day$ (approximately 1 simulated day per day) benchmark required by milestone-G.

3 Description of Algorithms

This section provides an overall description of the land surface modeling and data assimilation, followed by a description of the algorithms for each individual components of LIS involved in this code improvement study.

3.1 Land Surface Modeling and Data Assimilation

In general, land surface modeling seeks to predict the terrestrial water, energy and biogeochemical processes by solving the governing equations of the soil-vegetation-snowpack medium. Land surface data assimilation seeks to synthesize data and land surface models to improve our ability to predict and understand these processes. The ability to predict terrestrial water, energy and biogeochemical processes is critical for applications in weather and climate prediction, agricultural forecasting, water resources management, hazard mitigation and mobility assessment.

In order to predict water, energy and biogeochemical processes using (typically 1-D vertical) partial differential equations, land surface models require three types of inputs: 1) initial conditions, which describe the initial state of land surface; 2) boundary conditions, which describe both the upper (atmospheric) fluxes or states also known as "forcings" and the lower (soil) fluxes or states; and 3) parameters, which are a function of soil, vegetation, topography, etc., and are used to solve the governing equations.

3.2 LIS driver

The main driver in LIS is derived from the Land Data Assimilation System (LDAS) [5]). LIS is a model control and input/output system (consisting of a number of subroutines, modules written in Fortran 90 source code) that drives multiple offline one dimensional land surface models (LSMs) using a vegetation defined "tile" or "patch" approach to simulate sub-grid scale variability. The one-dimensional LSMs such as CLM and NOAH, which are subroutines of LIS, apply the governing equations of the physical processes of the soil-vegetation-snowpack medium. These land surface models aim to characterize the transfer of mass, energy, and momentum between a vegetated surface and the atmosphere.

LIS makes use of various satellite and ground based observation systems within a land data assimilation framework to produce optimal output fields of land surface states and fluxes. The LSM predictions are greatly improved through the use of a data assimilation environment such as the one provided by LIS. In addition to being forced with real time output from numerical prediction models and satellite and radar precipitation measurements, LIS derives model parameters from existing topography, vegetation and soil coverages. The model results are aggregated to various temporal and spatial scales, e.g., 3 hourly, 1/4°.

The LIS driver has been constantly improved for all the past milestones. For Milestone E, the original LDAS driver was used in the baselining results presented for Milestone E. The LIS driver used for demonstrating code improvements for Milestone H was developed by adopting the core LDAS driver and implementing code

improvements for enhancing performance. The structure of LIS driver was also redesigned using object-oriented principles, providing adaptable interfaces for ease of code development and extensibility. For a detailed description of the redesign and code improvements, please refer to the interoperability document.

Figure 1 shows the algorithmic steps involved in the LIS driver. The execution of LIS driver starts with reading in the user specifications. The user selects the model domain and spatial resolution, the duration and timestep of the run, the land surface model, the type of forcing from a list of model and observation based data sources, the number of "tiles" per grid square (described below), the soil parameterization scheme, reading and writing of restart files, output specifications, and the functioning of several other enhancements including elevation correction and data assimilation.

The system then reads the vegetation information and assigns subgrid tiles on which to run the one-dimensional simulations. LIS driver runs its 1-D land models on vegetation-based "tiles" to simulate variability below the scale of the model grid squares. A tile is not tied to a specific location within the grid square. Each tile represents the area covered by a given vegetation type.

Memory is dynamically allocated to the global variables, many of which exist within Fortran 90 modules. The model parameters are read and computed next. The time loop begins and forcing data is read, time/space interpolation is computed and modified as necessary. Forcing data is used to specify boundary conditions to the land surface model. The LSMs in the LIS driver are driven by atmospheric forcing data such as precipitation, radiation, wind speed, temperature, humidity, etc., from various sources. LIS driver applies spatial interpolation to convert forcing data to the appropriate resolution required by the model. Since the forcing data is read in at certain regular intervals, LIS driver also temporally interpolates time average or instantaneous data to that needed by the model at the current timestep. The selected model is run for a vector of "tiles", intermediate information is stored in modular arrays, and output and restart files are written at the specified output interval.

3.3 Community Land Model (CLM)

CLM is a 1-D land surface model, written in Fortran 90, developed by a grass-roots collaboration of scientists who have an interest in making a general land model available for public use. LIS currently uses CLM version 2.0, which was released in May 2002. The source code for CLM 2.0 is freely available from the National Center for Atmospheric Research (NCAR) (`http://www.cgd.ucar.edu/tss/clm/`). CLM is used as the land model for the community climate system model (CCSM) (`http://www.ccsm.ucar.edu/`) and the community atmosphere model (CAM) (`http://www.cgd.ucar.edu/cms/`). CLM is executed with all forcing, parameters, dimensioning, output routines, and coupling performed by an external driver of the user's design

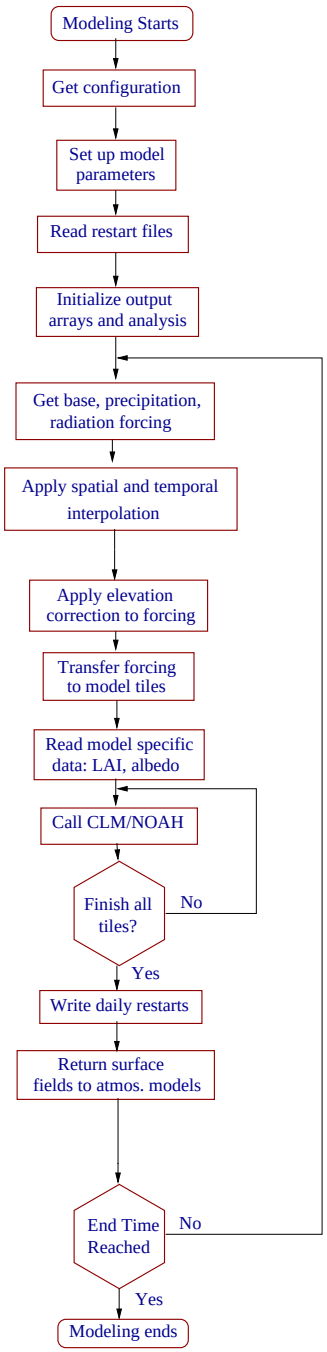

Figure 1: Flowchart for LIS driver

(in this case done by LIS driver). CLM requires pre-processed data such as the land

surface type, soil and vegetation parameters, model initialization, and atmospheric boundary conditions as input. The model applies finite-difference spatial discretization methods and a fully implicit time-integration scheme to numerically integrate the governing equations. The model subroutines apply the governing equations of the physical processes of the soil-vegetation-snowpack medium, including the surface energy balance equation, Richards' [10] equation for soil hydraulics, the diffusion equation for soil heat transfer, the energy-mass balance equation for the snowpack, and the Collatz et al. [3] formulation for the conductance of canopy transpiration.

3.4 The Community NOAH Land Surface Model

The community NOAH Land Surface Model is a stand-alone, uncoupled, 1-D column model freely available at the National Centers for Environmental Prediction (NCEP; `ftp://ftp.ncep.noaa.gov/pub/gcp/ldas/noahlsm/`). NOAH can be executed in either coupled or uncoupled mode. It has been coupled with the operational NCEP mesoscale Eta model [1] and its companion Eta Data Assimilation System (EDAS) [11], and the NCEP global Medium-Range Forecast model (MRF) and its companion Global Data Assimilation System (GDAS). When NOAH is executed in uncoupled mode, near-surface atmospheric forcing data (e.g., precipitation, radiation, wind speed, temperature, humidity) is required as input. NOAH simulates soil moisture (both liquid and frozen), soil temperature, skin temperature, snowpack depth, snowpack water equivalent, canopy water content, and the energy flux and water flux terms of the surface energy balance and surface water balance. The model applies finite-difference spatial discretization methods and a Crank-Nicholson time-integration scheme to numerically integrate the governing equations of the physical processes of the soil vegetation-snowpack medium, including the surface energy balance equation, Richards' [10] equation for soil hydraulics, the diffusion equation for soil heat transfer, the energy-mass balance equation for the snowpack, and the Jarvis [4] equation for the conductance of canopy transpiration.

3.5 Variable Infiltration Capacity (VIC) model

VIC (Variable Infiltration Capacity) is a macroscale, grid-based hydrological model from the University of Washington and Princeton University, which parameterizes the dominant hydro-meteorological processes taking place at the land surface - atmospheric interface. A mosaic representation of land surface cover, and sub-grid parameterizations for infiltration and the spatial variability of precipitation, account for sub-grid scale heterogeneities in key hydrological processes.

4 Code Improvements and Parallelization in LIS

4.1 Memory Utilization

Since Milestone F, we have been continuously improving the resource utilization of LIS, especially memory usage, which has a fundamental impact on the overall performance of LIS.

In addition to improving the memory usage profile in each subroutines as we have done in Milestone F, we have measured the memory usage as a function of execution time for LIS, and identified some sub-optimal memory usage patterns. Figures 2 and 3 show examples of the memory usage for CLM and NOAH before and after this code improvement. Before the improvement, both CLM and NOAH showed large fluctuations of memory usage as a function of runtime. We identified the cause of this pathological usage pattern and improved the code to fix this behavior. After the improvement, we eliminated the memory usage spikes and keep the memory usage steady and low over the runtime. This improvement reduced LIS' memory requirement by more than 30%.

We also reduced VIC code's memory usage by about 50% by converting all the double precision floats into single precision ones, as we do not see the extra accuracy from double precision is necessary for LIS code.

4.2 Spatial Interpolation and LSM Code Improvement

Our profiling results from Milestone E demonstrated the functions that are most time-consuming, thereby identifying the portions of our code-set that require our immediate attention. These functions were mainly the spatial interpolation, temporal interpolation, and the land surface model runs. The code improvements for Milestone F concentrated on improving the performance of these functions. For spatial interpolation, our code employs an external library called `ipolates` from NCEP. The source code of this library's routines were rewritten to improve their performance. The time interpolation and the land surface model runs were parallelized using the scheme described below.

Most of the community LSMs and the original LDAS driver code were not originally designed with an intent to be run at such high resolutions as 5km and 1km. As a result, the internal data structures in these models and driver consisted of large data structures leading to considerable memory requirements at 5km and higher resolutions. For example, a "NOAH" tile will be defined for every point in the tile-space. The amount of memory required for the whole NOAH data structure will scale linearly with increase in grid points, leading to unmanageable memory requirements.

Our original estimates of memory from the baselining results predicts approxi-

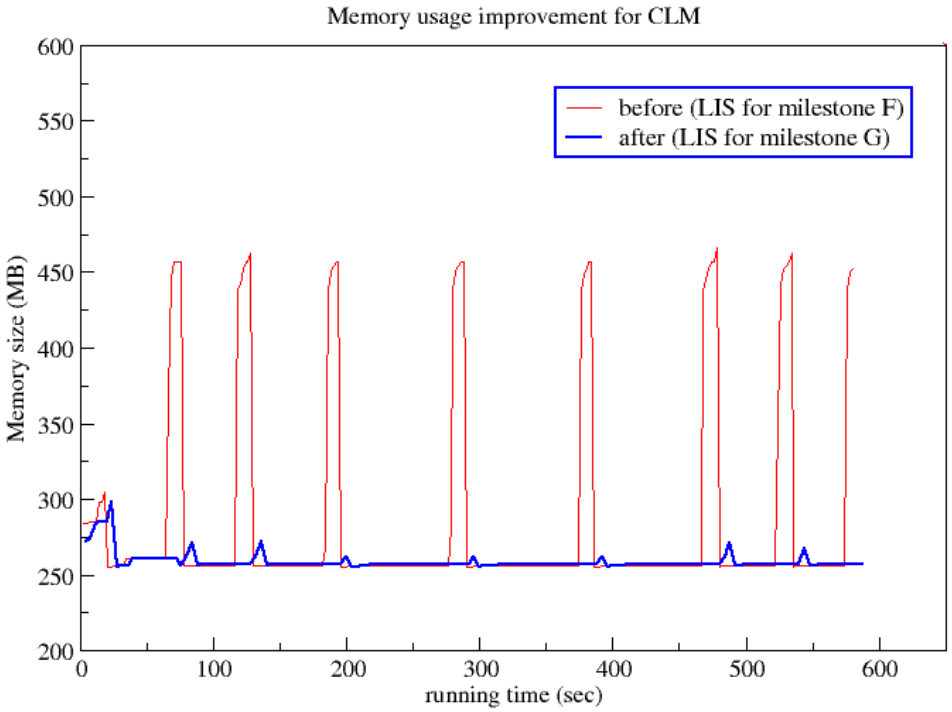

Figure 2: Memory usage improvement for CLM between LIS 2.0 for Milestone F and LIS 3.0 for Milestone G. This example shows the different memory usage patterns as a function of runtime for LIS running at regional 5km resolution.

mately 50GB and 90GB of memory for NOAH and CLM runs approximately. Our baselining code used an older version of CLM (Version 1.0). CLM version 2.0 is significantly more complex than version 1.0 in its functionality. For example, CLM version 1.0's and 2.0's main data structures contain approximately 450 and 550 variables, respectively. CLM version 2.0 contains a number of additional functionalities such as capability for coupling with atmosphere models, modules for river routing, volatile organic compounds emissions, dust mobilization analysis, etc. CLM2.0 includes extensive changes in surface datasets such as Leaf Area Index(LAI), Stem Area Index (SAI), canopy heights, percent clay and percent sand. CLM1.0 uses a single LAI value varying between a prescribed max/min value determined by soil temperature, whereas CLM2.0 uses a geographically and temporally varying value. In terms of the land surface physics, CLM2.0 includes calculation of orbital parameters and the plant

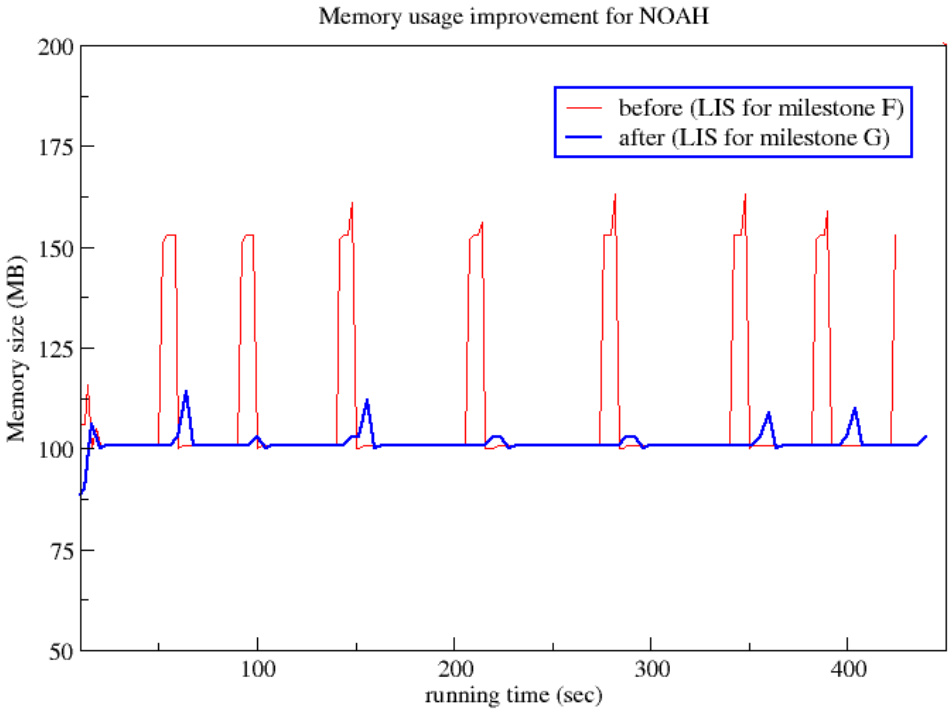

Figure 3: Memory usage improvement for NOAH between LIS 2.0 for Milestone F and LIS 3.0 for Milestone G. This example shows the different memory usage patterns as a function of runtime for LIS running at regional 5km resolution.

functional types determine the vegetation parameters, in addition to the sub-models described above.

We modified CLM 2.0 to improve the computational performance of the model and none of these changes alter the physics of the model. The list of modifications is shown below.

- CLM 2.0 in LIS uses the biogeophysics module only. The global data structure that defines the variables corresponding to each functional modules in CLM is modified to reflect this change. All the variables declared in the global data structure (clmtype.F90) corresponding to the following modules are commented out:
  - Biogeochemistry

  - Ecosystem dynamics
  - VOC model
  - Dust model
  - River routing model
  - Lake model

- CLM 2.0 contains NCAR's implementation of parallelization functions. These routines are eliminated, since the driver handles these functionalities in LIS. Further, routines to support parallel I/O using GDS Server are also built into the LIS version of the CLM code.

- Since the global data structure in CLM is mainly responsible for the amount of memory used when CLM is executed, the data structure is optimized by eliminating a number of variables that do not need to be stored globally. For example, some of the following variables are eliminated:

  - latdeg, londeg : Since we already carry around lat/lon values in radians, it is unnecessary to store the lat/lon values in degrees. The code is modified to calculate lat/lon values in degrees wherever necessary.
  - error checking variables : The variables corresponding to water and energy balance error checks are eliminated from the global data structure. The error checks are still performed in the code. However, the error values are not stored.
  - The biogeophysics routines are contained in mainly two routines, Biogeophys1 and Biogeophys2. To avoid passing the variables that are common to both routines, a number of variables were declared in the global data structure. These variables are eliminated and are made part of the argument lists in the appropriate subroutines.

- The I/O routines, (including restart reading and writing) are modified. The LIS driver's generic routines to write output in various formats are used. Similarly, the input routines are modified to accept the boundary conditions, forcing, and parameters provided by the LIS driver.

4.3 Code Structure

LIS consists of an integrated set of components and routines organized into several functional abstractions to facilitate development of land surface modeling applications. The components in LIS are implemented using object-oriented design principles. LIS integrates these components into a framework facilitating their effective use

in solving a variety of different problems. The design of LIS aims at providing extensible features that enable the reuse of LIS structure as well as the use of standards and tools proposed by other earth system modeling groups. The following sections describe these aspects of LIS design.

The entire code was redesigned, with an emphasis on reducing the size of the main data structures. The constructs in the LIS driver and the land surface model were significantly modified, including the vectorization scheme mentioned below, leading to considerable memory savings.

The global land surface is modeled by dividing it into two-dimensional regions or cells (e.g. cells of size 1km x 1km, which would lead to approximately 50,000 times more grid points than that of a simulation with cells of size $2^\circ \times 2.5^\circ$). Each cell can have a partial spatial coverage by a number of vegetation types, as well as bare soil. The vegetation characteristics such as leaf area index, stomatal resistance, etc. might be time varying. The conditions in each cell (energy, water fluxes, etc.) are computed at different time intervals. Each cell is driven by different atmospheric forcing variables. Assuming approximately 0.4 milliseconds per day for each LSM run on a particular cell, it can be estimated that modeling global land surface processes over a year with 15 minute timesteps on a 1km grid would require approximately 74 years of runtime on a single processor. This problem is clearly a grand challenge simply from computational perspective.

Land surface processes have rather weak horizontal coupling on short time and large space scales, which enables efficient scaling across massively parallel computational resources. LIS driver takes advantage of this weak horizontal coupling of land surface processes in the parallelization scheme described below.

4.4 Parallel Processing

To improve flexibility, portability and optimal utilization of computing resources, we have developed LIS to support two parallel processing schemes.

The first scheme is the conventional MPI-based parallel processing scheme, which is best suited to run LIS on multi-processor systems with sufficient memory and reliable nodes. With limited memory, this mode can perform low resolution global runs or high resolution regional runs, the scale of which is determined by the available memory.

Figure 4 shows the first parallelization scheme employed in LIS, first designed for Milestone F, and has ever since been implemented and improved. The program starts by initializing the global grid and associated parameters. Since land surface modeling is conducted only on the land points, the program switches to a vectorized representation of grid, which in turn leads to significant memory savings. For example, at 5km resolution, the number of grid points is approximately 21 million. By using

a vectorized grid representation, we conduct model runs on only approximately 6 million points. A dynamic domain decomposition based on the number of processors is carried out on the vectorized grid. The master processor conducts the spatial interpolation. Temporal interpolation and land surface model runs are carried out on the compute nodes based on their assigned subdomain. The master collects the variables required for output, and the loop continues until the simulation is complete.

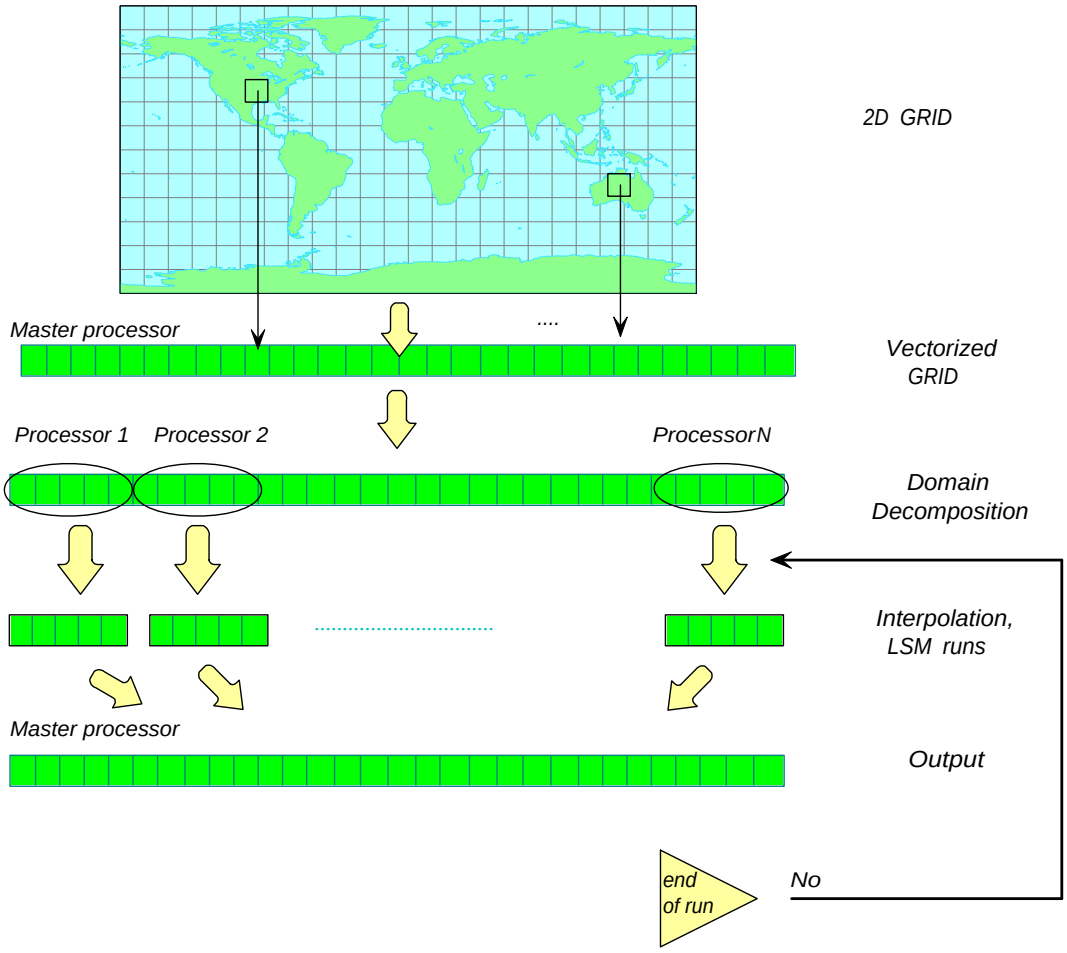

Figure 4: MPI-based Parallelization Scheme in LIS

The second scheme is based on the "pool of task" design, which takes advantage of the coarse-grained feature of LIS parallel processing, and is best suitable to perform high resolution, global LIS simulations on Beowulf clusters with limited memory. This scheme also distinguishes itself with strong fault-tolerance in the face of node crashes, thus lowering the hardware requirements for the compute nodes. Supported

by parallel and distributed input/output, this scheme for LIS can provide near-linear scalability, as we will show in the following sections.

Figure 5 shows the parallelization scheme used in the second code improvement for Milestone G, for LIS runs on our Beowulf cluster. The "pool of tasks" paradigm we are using is equivalent to a master–slave programming model, where we use one of the IO nodes as the master node and it distributes jobs to the slave (compute) nodes. We refer to the master node as "farmer" and the compute nodes as "dogs".

The master node ("farmer") will keep three tables on hand when starting the job: table of unfinished-jobs ("bones"), finished-jobs ("done"), and jobs-fetched ("munching"). At the beginning, all the jobs are stored in the "bones" table. Each compute node ("dog") sends a request to the master to request a job from it, and starts working on it when a job is assigned by the master node. The master node then moves the fetched jobs to the "munching" table, and starts a timer for each fetched job. The timer specification will be based on the estimation of the execution time a compute node needs to finish a job. When a compute node finishes a job and notifies the master node before the job's corresponding timer runs out, this piece is regarded a finished job, and the master node moves it from the "munching" table to the "done" table. And the compute node goes on to request another job until the "bones" table is empty.

The fault–tolerance is realized with a time–out mechanism based on the timer the master node keeps. If the timer of a fetched job runs out before the compute node reports back, the master node then assumes that that particular compute node must have crashed, and then moves that timed-out job from the "munching" table back to the "bones" table for other compute nodes to fetch. The flow chart in Figure 5 shows the master node's job handling and scheduling process (left), and the various states of the three tables (right) the master node uses to keep track of the job progress at different corresponding stages in the flow chart.

This farmer-dog system maximize resource utilization even when the compute nodes ("dogs") have heterogeneous memory/CPU configuration. Each job ("bone") naturally has different number of total land points, depending on its location on the Earth. The job management system will make sure all the nodes will get to work on the biggest jobs they can handle first, depending on their hardware configuration. Only after they finish the biggest jobs can they continue to work on smaller jobs. This scheme ensures the more powerful nodes are not fighting for small jobs with less powerful nodes before they finish the jobs only they can handle. This has proven to be working well on the LIS cluster, since about half of the computer nodes have 1GB memory while the other half have only 0.5GB.

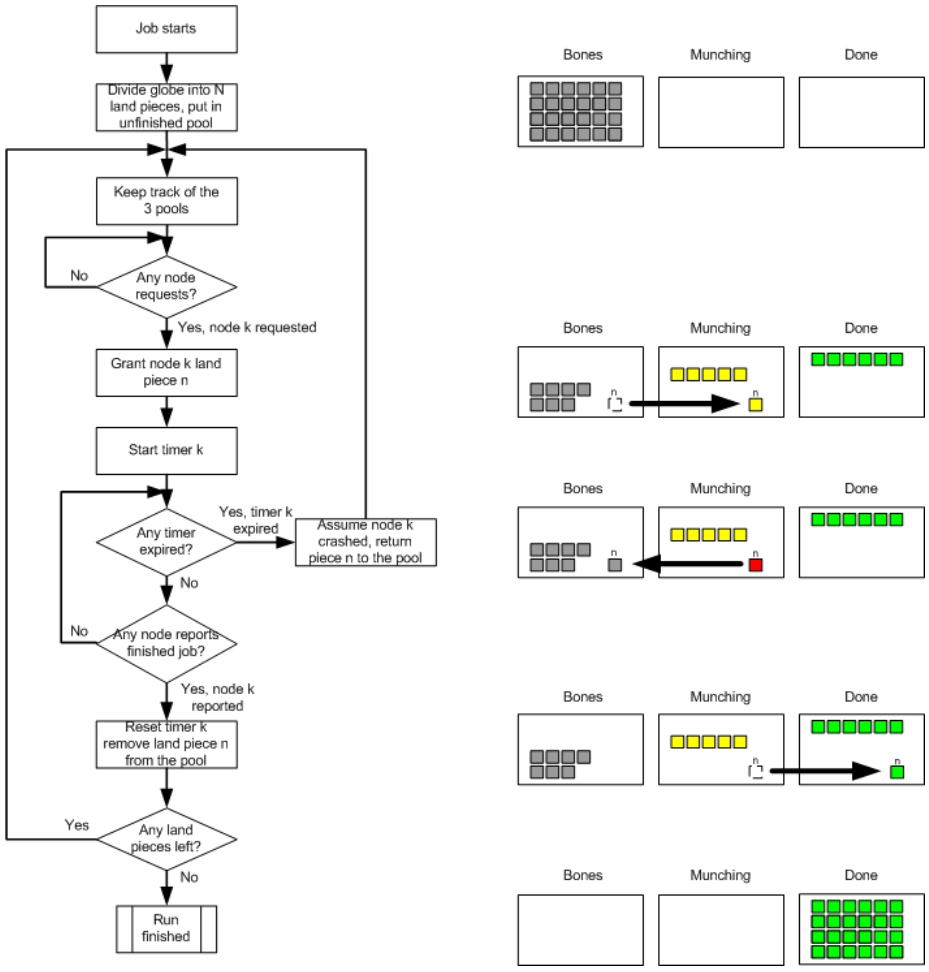

Figure 5: Task-pool-based Parallelization Scheme in LIS. Left panel shows the logic of the master node. Right panel shows the movement of jobs in the three pools: bones, munching and done. Each job in the "munching" pool is timed so when a job is timed out (red block), it will be put back to the "bones" pool for other nodes to handle. Time-out happens when a compute node crashes.

4.5 Parallel and Distributed Input/Output

To maintain scalability when the number of processors to be used for this milestone is high (200 nodes), we have to ensure the input/output performance imposes as little bottleneck as possible.

For input, the LIS code now has the capability to retrieve data from a pool of many GrADS-DODS (GDS) servers running in parallel with mirrored data. We have implemented a transparent load–balancing mechanism which can dynamically assign

data serving requests to GDS servers based on their load. Servers can be added or removed from the server pool dynamically without interrupting ongoing simulations. This implementation not only increases total input throughput, but also ensures high availability of the data service.

For output, we have made a fundamental improvement to the GDS code to make it support distributed data storage, similar to the PVFS design. The LIS output data now do not have to be gathered and assembled onto one single disk for GDS to serve; instead, GDS can serve data directly off the compute nodes' local disks. This improvement eliminates the process for the compute nodes to send back the output data to the IO node, which has been a bottleneck. With this new GDS capability, the compute nodes can now write the output data to their local disks with much faster speed and less overhead. This architecture is very similar to the Parallel Virtual File System (PVFS) [9], with our customized data striping scheme that best fits the LIS data output pattern.

5 Description of the Test

5.1 Test platform

All the benchmarking runs were performed on the LIS Linux cluster, constructed at Milestone-f. The cluster has 200 computers in total. Eight of them are the IO nodes, each having dual AMD XP 2000 CPUs, 2 GB DDR memory, 220 GB internal disk storage, 2 built-in Ultra-SCSI channels, 2 built-in fast Ethernet adapters, and 1 gigabit Ethernet adapter (two of the IO nodes have two gigabit adapters each). Meanwhile, each IO node has one external Promise RAID systems for file storage, with a capacity of 1 TB in each RAID system. The remaining 192 computers are compute nodes. Each of them has an AMD XP 1800 CPU, 512 or 1024 MB memory, an internal IDE hard drive of 81 GB, and a built-in fast Ethernet adapter.

Overall, the LIS cluster has 208 AMD XP processors of 1.53 GHz and above, 160 GB of memory, 24 TB of disk space, 192 fast Ethernet connections, and 10 gigabit Ethernet connections.

We are using MPICH-1.2.4 for message passing interface (MPI) support, and Absoft ProFortran 8.0 as our F90 compiler.

5.2 Test domain and configuration

All the tests were performed on the global domain with 1-km resolution (1/100 degree). A time step of 30 minutes was used for all the runs, and the data output interval is 3 simulated hours. For simplicity, only one tile per grid was simulated in

the runs. The output files were written using binary format.

The following parameters were varied for different test configurations:

- Land surface models: NOAH, CLM or VIC;
- Base forcing: GDAS or GEOS;
- Number of compute nodes: 32, 64, 128 or higher. Sixteen (16) nodes were used occasionally;
- Number of GrADS-DODS servers: 2, 3, 4, 5, or 6;

5.3 Description of the test code

We used LIS 3.0 code for the tests. The LIS code is composed of a driver and the three land surface models (LSMs): NOAH, CLM and VIC. LIS driver controls the LSM models and input/output operations (consisting of a number of subroutines, modules written in Fortran 90 source code). It drives these offline one-dimensional LSMs using a vegetation defined "tile" or "patch" approach to simulate subgrid scale variability. These one-dimensional LSMs, which are subroutines of the LIS driver, apply the governing equations of the physical processes of the soil-vegetation-snowpack medium to simulate the Earth's surface conditions.

For this milestone, the versions of the three LSMs are: CLM 2.0, NOAH 2.6 and VIC 4.0.3.

5.4 Documentation and repository of the test code

The detailed documentation and source code of the LIS driver 3.0 and the land surface models (CLM 2.0, NOAH 2.6 and VIC 4.0.3) can be accessed at `http://lis.gsfc.nasa.gov/Documentation/LIS3.0/index.html`

6 Results

This section gives detailed documentation of the test results for milestone G. We will demonstrate that LIS code is running on the cluster with nice scalability, and the whole system consistently and reliably operates at 1km resolution with performance much better than $0.4ms/cell \cdot day$ as specified by milestone G.

The LIS driver was running on LIS cluster with various combinations of LSM, base-forcing and observed forcing. In addition, we performed performance benchmarking on the cluster with a varying number of compute nodes to show the scalability of the code. Finally, we also studied the impact of the number of GrADS-DODS servers

on the performance. The simulated period of time considered for all the runs in this study is 1 day, the same as the previous milestones.

6.1 LIS with NOAH and base forcing

Figure 6 shows the performance and scaling test results for NOAH with either GDAS or GEOS base forcing. Estimated P/2 scaling curves are also shown as dashed lines.

As the figure shows, with 32 compute nodes and up, LIS can run with NOAH with a normalized of timing of lower than $0.15ms/grid \cdot day$. With 128 compute nodes, NOAH with GEOS ran at approximately $0.08ms/grid \cdot day$, while NOAH with GDAS at $0.05ms/grid \cdot day$. NOAH with GEOS reached $0.075ms/grid \cdot day$ with 170 nodes, and NOAH with GDAS hit $0.045ms/grid \cdot day$ with 197 nodes. All the timing results shown are much better than $0.4ms/grid \cdot day$.

To see how realistic for LIS to perform near-realtime land surface simulations, the figure also illustrates the simulation throughput as shown by the numbers on the right y-axis. With 32 nodes, both NOAH/GEOS and NOAH/GDAS runs can finish 2.5 simulation days in 1 day, while with 170 compute nodes, NOAH/GEOS ran at a speed of 4.5 simulated days/day, while NOAH/GDAS 7.5 days/day with 197 nodes. Therefore, LIS is well positioned to perform near-realtime runs.

For both cases, the timing kept decreasing as the number of compute node was increased, with a slight tendency of leveling only for more than 128 nodes. For NOAH/GEOS, the scaling is better than P/2 curve for 32 and 64 nodes, and NOAH/GDAS is better than P/2 even with 128 compute nodes. The trend of leveling off with 128 nodes and more indicates the performance bottleneck is being shifted to the GDS servers, which have longer delays when serving data to more clients. In addition, NOAH is the least CPU-intensive model among the three LSMs, and it is expected that the performance bottleneck shifts from CPU to IO (GDS serving) sooner than CLM and VIC when the number of compute nodes increases. This could explain the worse-than-P/2 scaling with more than 128 nodes for NOAH/GEOS, with the fact that GEOS data serving is slower than GDAS, as shown in Sec. 6.4.

6.2 LIS with CLM and base forcing

The test results for CLM with either GDAS or GEOS base forcing are shown in Fig. 7, together with the estimated P/2 scaling curves shown as dashed lines.

As the figure shows, CLM/GEOS with 16 nodes ran at approximately $0.35ms/grid \cdot day$, which is only slightly faster than the milestone $0.4ms/grid \cdot day$. However, with 32 nodes, CLM/GEOS timed $0.2ms/grid \cdot day$, and CLM/GDAS scored $0.27ms/grid \cdot day$. The timing decreased significantly as the number of compute node was increased, and CLM/GEOS ran at $0.076ms/grid \cdot day$ with 152 nodes, and CLM/GDAS

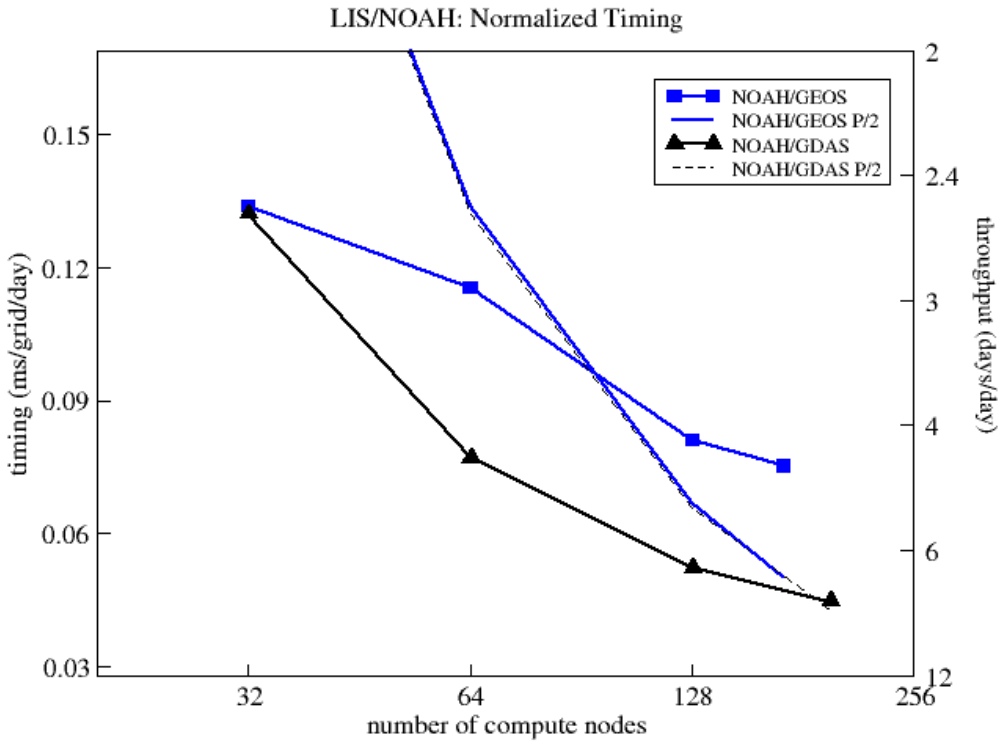

Figure 6: Normalized timing (left y-axis) and simulation throughput (right y-axis) for LIS with NOAH LSM and the two base forcing settings. Five (5) GDS servers were used. The P/2 timing curve is based on 1-node timing extrapolated from 32-node timing. The maximum number of compute nodes used for NOAH/GEOS is 170, and for NOAH/GDAS 197.

$0.05 ms/grid \cdot day$ with 164 nodes, corresponding to throughputs of 4.4 days/day and 6.7 days/day, respectively.

CLM shows better scaling than P/2 curve up to 128 compute nodes. CLM/GDAS still performed better than P/2 even with 164 nodes.

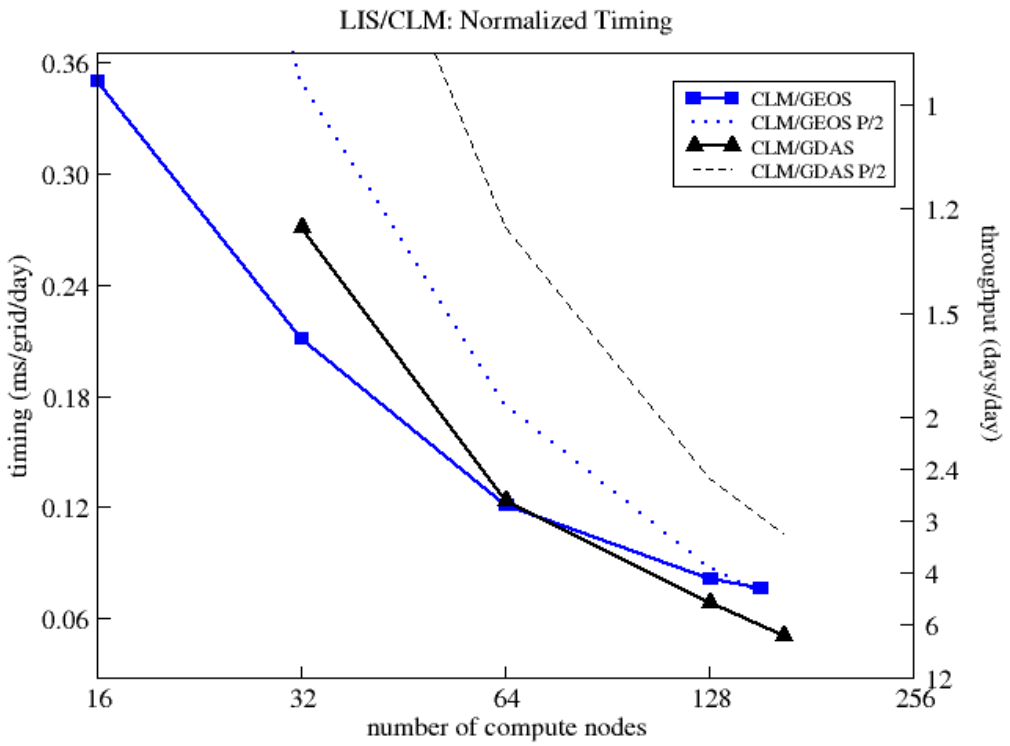

Figure 7: Normalized timing (left y-axis) and simulation throughput (right y-axis) for LIS with CLM LSM and the two base forcing settings. Five (5) GDS servers were used. The P/2 timing curve is based on 1-node timing extrapolated from 16-node timing for CLM/GEOS, and 32-node timing for CLM/GDAS. The maximum number of compute nodes used for CLM/GEOS is 152, and for CLM/GDAS 164.

6.3 LIS with VIC and base forcing

The performance results for VIC with either GDAS or GEOS base forcing are shown in Fig. 8, with the corresponding estimated P/2 scaling curves shown as dashed lines.

The performance of VIC with the improved LIS code is very close to that of CLM and NOAH. With 32 compute nodes, VIC with GEOS ran at approximately $0.3ms/grid \cdot day$, which is only slightly faster than the milestone $0.4ms/grid \cdot day$. VIC with GDAS ran nearly twice as fast with the same number of compute nodes.

VIC performed much better than the targeted $0.4ms/grid \cdot day$ when 64 and more compute nodes were used. With 64 compute nodes, VIC/GEOS ran at about $0.1ms/grid \cdot day$, which is a throughput of more than 3 simulated days per day. VIC/GDAS ran at $0.08ms/grid \cdot day$, or more than 4 days/day. With 128 nodes, VIC/GDAS' performance kept increasing substantially, while VIC/GEOS started to show slight leveling. With 163 nodes, VIC/GDAS can run at $0.04ms/grid \cdot day$, which is about 10 days/day. However, VIC/GEOS' performance degraded with 161 nodes, showing a lower throughput than 128 nodes. Further tests will be needed to identify the cause of this performance drop, and we conjecture this may be due to some unusual system load by other jobs running on the cluster at the time of the test.

Both VIC/GDAS and VIC/GEOS showed better performance than their corresponding P/2 scaling curve for all the tests. However, they showed the tendency of leveling off as the number of computer nodes was above 160.

6.4 Inter-comparison of LSMs and base forcing

With the improved LIS code, the performances for all the three LSMS (NOAH, CLM and VIC) are fairly close, as shown in Fig. 9. With 128 compute nodes and 5 GDS servers, all the 3 LSMs can run LIS at lower than $0.1ms/grid{\cdot}day$, or with throughputs better than 3.4 days/day.

With all the three LSMs, GDAS forcing is systematically faster than GEOS forcing. For example, VIC/GDAS was running as about twice fast as VIC/GEOS. The performance difference for CLM between GEOS and GDAS forcing is the smallest.

6.5 Scaling tests of GrADS-DODS servers

For LIS runs, we used multiple GDS servers running in parallel with dynamic load balancing. To demonstrate how this setting affects LIS performance, we performed tests with 2, 3, 4, 5, and 6 GDS servers. The timing results for NOAH/GEOS runs with 128 compute nodes are shown in Fig. 10.

As the figure shows, the performance improved as the number of GDS servers was increased. With 2 GDS servers, NOAH/GEOS ran at $0.122ms/grid \cdot day$, while with 6 GDS servers, it ran at approximately $0.078ms/grid \cdot day$, with a throughput improvement from 2.8 days/day to 4.4 days/day. However, the most significant performance improvement took place when the number of GDS servers was increased from 2 to 3. More GDS servers helped, but not as much. It is reasonable to expect that more GDS servers (4, 5, and 6) will be more helpful for runs with when the number of compute nodes is further increased.

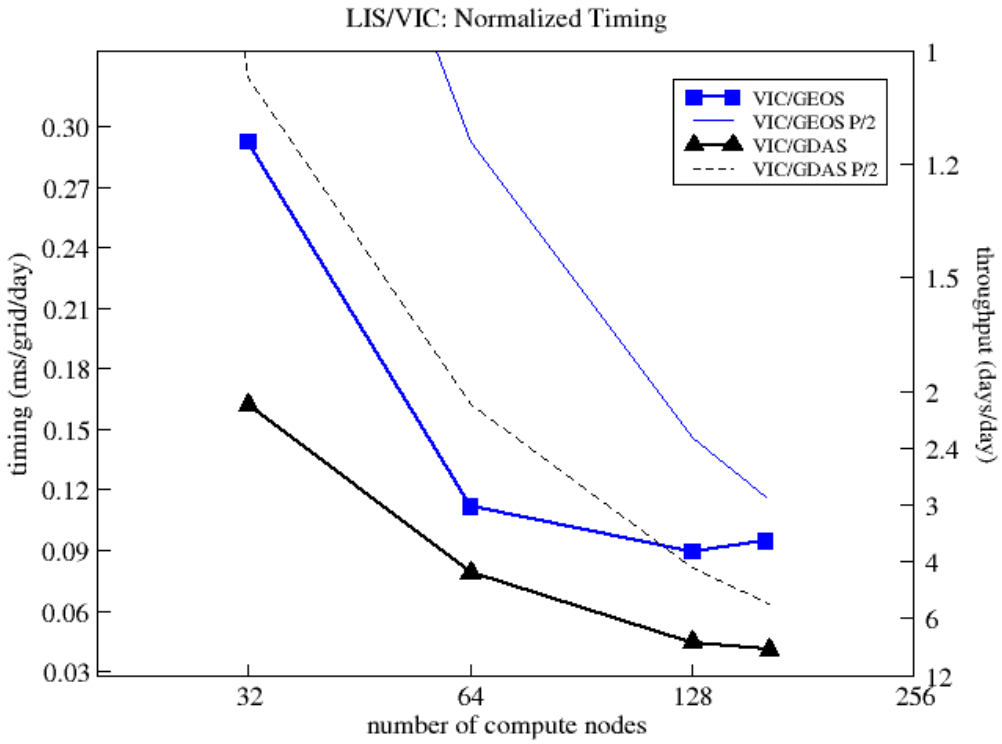

Figure 8: Normalized timing (left y-axis) and simulation throughput (right y-axis) for LIS with VIC LSM and the two base forcing settings. Five (5) GDS servers were used. The P/2 timing curve is based on 1-node timing extrapolated from 32-node timing. The maximum number of compute nodes used for VIC/GEOS is 161, and for VIC/GDAS 163.

7 Final Remarks

The LIS code, now LIS 3.0, has enjoyed significant improvement since last code improvement milestone. Compared to previous versions of LIS code, LIS 3.0 is cleaner, more efficient, more portable, and most importantly, faster.

The code is more structured and cleaner, with a more modular architecture and separation of functionalities. This enables easy integration of other LSMs into LIS code. As a matter of fact, the LSM Mosaic has been successfully integrated in LIS

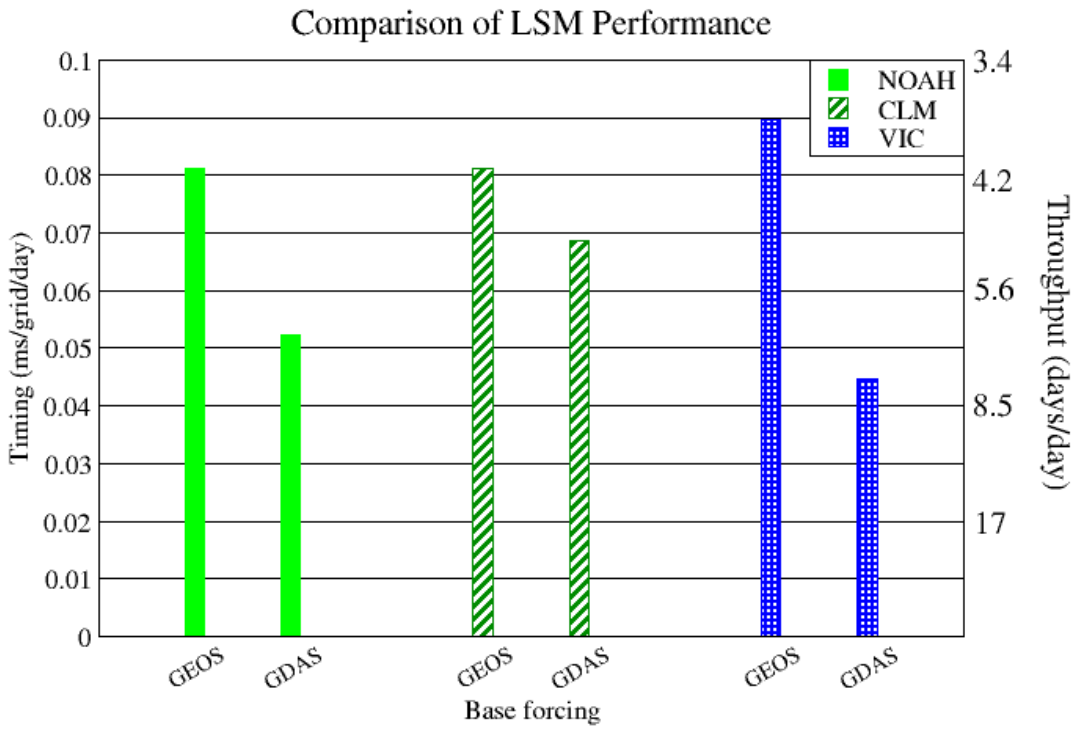

Figure 9: Performance comparison between the three LSMs: CLM, NOAH and VIC. Each LSM was tested with both GEOS and GDAS forcing. Five (5) GDS servers and 128 compute nodes were used for all the tests.

by LIS users.

The overall memory requirement has dropped by 30% or more, with a steady memory usage pattern. One core component, the spatial interpolation routine, has been greatly optimized with faster speed and less memory usage.

In addition, LIS 3.0 now has greater flexibility and portability with two modes of parallel computation: MPI based, synchronized parallelization and "pool of tasks", coarse-grained parallelization, to suit different platforms and hardware configurations, and give users greater control. In particular, LIS 3.0 now can run with support of parallel GDS servers and perform distributed output, paving the way for significant

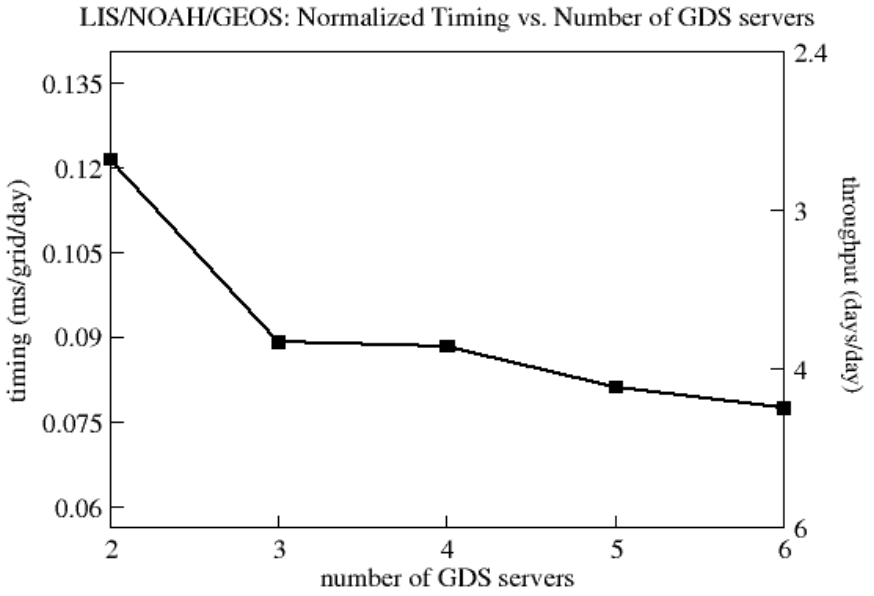

Figure 10: Normalized timing (left y-axis) and simulation throughput (right y-axis) as functions of the number of GDS servers used, for LIS with NOAH LSM, GEOS base forcing and 128 compute nodes.

performance improvement.

All these improvements made it possible for LIS to run global land simulations with 1km resolution successfully. Most importantly, the performance of LIS met and exceeded the requirement of this milestone, making the claim of "one day in less a day" a reality.

References

[1] F. Chen, K. Mitchell, J. Schaake, Y. Xue, H. Pan, V. Koren, Y. Duan, M. Ek, and A. Betts. Modeling of land-surface evaporation by four schemes and comparison with fife observations. *J. Geophys. Res.*, 101(D3):7251–7268, 1996.

[2] CLM. http://www.cgd.ucar.edu/tss/clm/.

[3] G. J. Collatz, C. Grivet, J. T. Ball, and J. A. Berry. Physiological and environmental regulation of stomatal conducatance: Photosynthesis and transpiration: A model that includes a laminar boundary layer. *Agric. For. Meteorol.*, 5:107–136, 1991.

[4] P. G. Jarvis. The interpretation of leaf water potential and stomatal conductance found in canopies of the field. *Phil. Trans. R. Soc.*, B(273):593–610, 1976.

[5] LDAS. http://ldas.gsfc.nasa.gov.

[6] LIS. http://lis.gsfc.nasa.gov/.

[7] NOAH. ftp://ftp.ncep.noaa.gov/pub/gcp/ldas/noahlsm/.

[8] C. D. Peters-Lidard, S. Kumar, Y. Tian, J. L. Eastman, and P. Houser. Global urban-scale land-atmosphere modeling with the land information system. *Symposium on Planning, Nowcasting, and Forecasting in the Urban Zone, 84th AMS Annual Meeting 11-15 January 2004 Seattle, WA, USA.*, 2004.

[9] PVFS. http://www.parl.clemson.edu/pvfs/.

[10] L. A. Richards. Capillary conduction of liquids in porous media. *Physics*, 1:318–333, 1931.

[11] E. Rogers, T. L. Black, D. G. Deaven, G. J. DiMego, Q. Zhao, M. Baldwin, N. W. Junker, and Y. Lin. Changes to the operational "early" eta analysis/forecast system at the national centers of environmental prediction. *Wea. Forecasting*, 11:391–413, 1996.

[12] VIC. http://hydrology.princeton.edu/research/lis/index.html.